REMARKS

The above Amendments and these Remarks are in reply to the Office Action mailed April 19, 2007.

Currently, claims 1-37 are pending. Applicants respectfully request reconsideration of claims 1-37.

I. Rejection of Claims 1-5, 7, 13-15, 18, 19, 21, 26, 27, 29, and 35 Under 35 U.S.C. §102(b)

Claims 1-5, 7, 13-15, 18, 19, 21, 26, 27, 29, and 35 have been rejected under 35 U.S.C. §102(b) as being anticipated by Steele. Because the cited prior art does not disclose all of the limitations of claims 1-5, 7, 13-15, 18, 19, 21, 26, 27, 29, and 35, Applicants respectfully assert that all of the claims are patentable over the cited prior art.

As discussed in the previous response to an Office Action dated 10/31/2006, Steele does not disclose code that "includes one or more expressions and one or more markers that specify when said one or more expressions should be evaluated during execution of said program," as recited in claim 1. Instead, Steele discloses the `eval-when` special form used in Common Lisp, which allows evaluation of the body of the form "only at compile time, only at load time, or when interpreted but not compiled" (Steele, p. 11, lines 8-9). Each of these situations is indicated by the symbols `compile`, `load`, and `eval`, respectively, and indicates whether the compiler or the interpreter should evaluate the body. However, the `eval-when` form does not indicate when to evaluate the body "during execution" of the form. Instead, the `eval-when` form indicates which component should evaluate the body of the form (compiler or interpreter).

On page 21 of the Office Action, the Examiner argues that the `eval` situation, which evaluates the body of the `eval-when` form "when interpreted but not compiled" (Steele, p. 11, lines 8-9), occurs during execution time. However, the `eval` situation is simply used to specify that the body of the `eval-when` form should be evaluated **at** execution time, as opposed to either compile time or load time. It specifies that the interpreter **will** evaluate the body, but it

does not specify **when** the interpreter should evaluate the body during execution. Therefore, Steele does not disclose the claimed feature, as suggested by the Examiner.

The "markers that specify when said one or more expressions should be evaluated **during** execution," described in claim 1, are used to indicate a time of evaluation **while a program is running**. During execution of a program, an expression can be evaluated at the time specified by its marker, independent of where the expression is syntactically located. For example, as described in the Specification, the "markers" can indicate a time of `immediately`, `once`, or `always`. The "markers" can also specify a particular time such as `once@2s`, which would evaluate the expressions two seconds after execution begins (see Specification, p. 8-9, paragraph 0029). The `eval-when` form does not perform this function. The form disclosed in Steele does not specify a time to evaluate the expression "**during execution**." Instead, the form simply specifies the component (compiler or interpreter) that should evaluate the "expression."

As Steele explains, the `eval-when` form is used to specify what component should process the body (compiler or interpreter). There are also two modes in Common Lisp that are used to indicate which component should process the form, `not-compile-time` mode (interpreter) or `compile-time-too` mode (compiler). "The `eval-when` special form controls which of these two modes to use" (Steele, p. 11, lines 21). "The `eval-when` special form is used to annotate a program in a way that allows the program doing the processing to select the appropriate mode" (Steele, p. 12, line 27-28 and table on page 13).

To further clarify what the eval-when form does, below is an example taken from an online document entitled Common Lisp Extensions generated by DJ Delorie on June 23, 2003 (http://www.delorie.com/gnu/docs/emacs/cl_9.html):

Some simple examples:

```
;;  Top-level forms in foo.el:
(eval-when (compile)                (setq  foo1  'bar))
(eval-when (load)                   (setq  foo2  'bar))
(eval-when (compile load)           (setq  foo3  'bar))
(eval-when (eval)                   (setq  foo4  'bar))
(eval-when (eval compile)           (setq  foo5  'bar))
(eval-when (eval load)              (setq  foo6  'bar))
(eval-when (eval compile load)      (setq  foo7  'bar))
```

When '`foo.el`' is compiled, these variables will be set during the compilation itself:

```
foo1  foo3  foo5  foo7  ;     `compile'
```

When '`foo.el`' is loaded, these variables will be set:

```
foo2  foo3  foo6  foo7  ;     `load'
```

When '`foo.el`' is loaded uncompiled, these variables will be set:

```
foo4  foo5  foo6  foo7  ;     `eval'
```

As shown above, `compile`, `load`, and `eval` simply indicate which component evaluates the body of the `eval-when` form (at compile time, at load time, or at execution time). It does not, however, indicate when "**during execution**" to evaluate the form.

Typically, the `eval-when` form is used in writing macros, where it is useful to evaluate a form within the `defmacro` body at i.e. compile time in order to keep the behavior of the macro consistent when the program is executed. The `eval-when` form allows for control of evaluation of the body.

> For example, if a file contains (`setq foo t`), the act of compiling it will not actually set `foo` to `t`. This is true even if the `setq` was a top-level form (i.e., not enclosed in a `defun` or other form). Sometimes, though, you would like to have certain top-level forms evaluated at compile-time. For example, the compiler effectively evaluates `defmacro` forms at compile-time so that later parts of the file can refer to the macros that are defined (Delorie p. 1).

The purpose of the `eval-when` form is to indicate whether its body should be processed at compile, at load, or at execution time. However, the `eval-when` form does not specify when "**during execution**" to evaluate the body. It only specifies what component (compiler or interpreter) should evaluate the body.

Because Steele does not disclose code that "includes one or more expressions and one or more markers that specify when said one or more expressions should be evaluated during

execution of said program," the reference does not anticipate claim 1, as argued by the Examiner. The `eval-when` form does not in fact specify when "**during** execution" to evaluate the body of the form. Claims 2-5, 7, 13-15, 18, 19, 21, 26, 27, 29, and 35 are distinguishable over the cited prior art for at least the same reasons as claim 1. Applicants respectfully request reconsideration of these claims.

Additionally, claim 3 discloses that the "markers can indicate that a particular expression should be evaluated immediately, once, or always." The Examiner argues that the situations disclosed in Steele (`compile`, `load`, or `eval`) can be equated to "immediately, once, or always." However, this is not the case. Immediately, once, and always indicates a time **during** execution to evaluate an expression, as described in the Specification:

> If the "time" is indicated to be "immediately," then the system will initialize the attribute to the value of expression when the enclosing element is defined... If "time" indicates "once," then the system initializes the attribute to the value of expression when the enclosing element is initialized (or instantiated)... If "time" indicates "always," then the system updates the attribute any time the value of the expression changes. That is, the attribute is said to be constrained to the value of expression... At each relevant time period, the appropriate expressions are evaluated (Specification, p. 8-9, paragraphs 0029-0030).

The situations for the `eval-when` form (`compile`, `load`, and `eval`) do not perform like immediately, once, and always. They simply specify that an expression should be evaluated at compile time, load time, or at execution time. Because Steele does not disclose "markers [that] can indicate that a particular expression should be evaluated immediately, once, or always," the reference does not anticipate claim 3. Applicants respectfully assert that claim 3 is patentable over the cited prior art for at least this reason. Reconsideration of this claim is respectfully requested.

II. Rejection of Claims 6, 8-12, 16, 17, 20, 22-25, 28, 30-34, 36, and 37 Under 35 U.S.C. 103(a)

A. Steele in view of Rodriguez

Claims 6, 8, 16, 30, and 36 have been rejected under 35 U.S.C. 103(a) as being unpatentable over Steele in view of Rodriguez. Because the cited prior art, alone or in combination, does not teach or suggest all of the limitations of the rejected claims, Applicants assert that the claims are in condition for allowance.

Steele, as discussed above, does not disclose "one or more expressions and one or more markers that specify when said one or more expressions should be evaluated during execution of said program," as recited in claim 1. Claims 6, 8, 16, 30, and 36 all contain a similar feature. Additionally, Rodriguez does not teach or suggest this feature. Instead, Rodriguez discusses using Lisp scripts within HTML code for interactive applications over the Internet, yet no such "markers" are disclosed. Therefore, the combination of Steele and Rodriguez does not disclose, teach, or suggest all of the limitations of claims 6, 8, 16, 30, and 36. Applicants respectfully request reconsideration of these claims.

B. Steele

Claims 9-12, 22, 23, 25, 31, 32, 34 have been rejected under 35 U.S.C. 103(a) as being obvious over the Steele reference. Because the cited prior art does not disclose, teach, or suggest all of the limitations of the rejected claims, Applicants assert that the claims are in condition for allowance.

Steele, as discussed above, does not disclose "one or more expressions and one or more markers that specify when said one or more expressions should be evaluated during execution of said program," as recited in claim 1. Claims 9-12, 22, 23, 25, 31, 32, 34 all include a similar feature. Also, Steele does not teach or suggest this limitation. Therefore, claims 9-12, 22, 23, 25, 31, 32, 34 are not obvious over Steele. Applicants respectfully request reconsideration of these claims.

C. Steele in view of Hickey

Claim 17 has been rejected under 35 U.S.C. 103(a) as being unpatentable over Steele in view of Hickey. Because the cited prior art, alone or in combination, does not teach or suggest all of the limitations of the rejected claims, Applicants assert that the claims are in condition for allowance.

Steele, as discussed above, does not disclose "one or more expressions and one or more markers that specify when said one or more expressions should be evaluated during execution of said program," as recited in claim 1. Claim 17 contains a similar feature. Additionally, Hickey does not teach or suggest this feature. Instead, Hickey discloses Lisp as a tool for web programming, yet no such "markers" are disclosed. Therefore, the combination of Steele and Hickey does not disclose, teach, or suggest all of the limitations of claim 17. Applicants respectfully request reconsideration of this claim.

D. Steele in view of Haible

Claims 20, 24, 28, and 33 have been rejected under 35 U.S.C. 103(a) as being unpatentable over Steele in view of Haible. Because the cited prior art, alone or in combination, does not teach or suggest all of the limitations of the rejected claims, Applicants assert that the claims are in condition for allowance.

Steele, as discussed above, does not disclose "one or more expressions and one or more markers that specify when said one or more expressions should be evaluated during execution of said program," as recited in claim 1. Claims 20, 24, 28, and 33 all contain a similar feature. Instead, Haible discloses implementations for Common Lisp, yet no such "markers" are disclosed. Therefore, the combination of Steele and Haible does not disclose, teach, or suggest all of the limitations of claims 20, 24, 28, and 33. Applicants respectfully request reconsideration of these claims.

E. Steele in view of Rodriguez and Haible

Claim 37 has been rejected under 35 U.S.C. 103(a) as being unpatentable over Steele in view of Rodriguez and further in view of Haible. Because the cited prior art, alone or in combination, does not teach or suggest all of the limitations of the rejected claims, Applicants assert that the claims are in condition for allowance.

Steele, Rodriguez, and Haible, as discussed above, do not disclose "one or more expressions and one or more markers that specify when said one or more expressions should be evaluated during execution of said program," as recited in claim 1. Claim 37 contains a similar feature. Therefore, the combination of Steele, Rodriguez, and Haible do not disclose, teach, or suggest all of the limitations of claim 37. Applicants respectfully request reconsideration of these claims.

Based on the above amendments and these remarks, reconsideration of claims 1-37 is respectfully requested.

The Examiner's prompt attention to this matter is greatly appreciated. Should further questions remain, the Examiner is invited to contact the undersigned agent by telephone.

The Commissioner is authorized to charge any underpayment or credit any overpayment to Deposit Account No. 501826 for any matter in connection with this response, including any fee for extension of time, which may be required.

Respectfully submitted,

Date: July 3, 2007 By: /Michelle Esteban/
Michelle Esteban
Reg. No. 59,880

VIERRA MAGEN MARCUS & DENIRO LLP
575 Market Street, Suite 2500
San Francisco, California 94105-4206
Telephone: (415) 369-9660
Facsimile: (415) 369-9665